Seminario de Computadores I,
1er Semestre 2004

Driver
Medidor de Distancia Ultrasónico

Francisco Blancaire
Eduardo González
Michael Kusch
Diego Valencia

Índice General

1 Introducción

En este documento se detalla la implementación de un driver para el módulo medidor de distancia ultrasónico, utilizando el microcontrolador MSP430F149.

Incluye una aplicación para la tarjeta de desarrollo Easy Web de Olimex, que lo utiliza para mostrar la distancia medida por el display LCD provisto.

2 Descripción del Módulo Ultrasónico

El módulo ultrasónico usado para medir distancia, correspondiente al SRF4 de Devantech, es descrito a continuación.

La medición se realiza emitiendo una onda sonora de alta frecuencia (40[KHz]) por un buzzer ultrasónico, luego se registra el tiempo que se demora en recibir un eco de esta onda al chocar con algún objeto, como se muestra en la Figura 1.

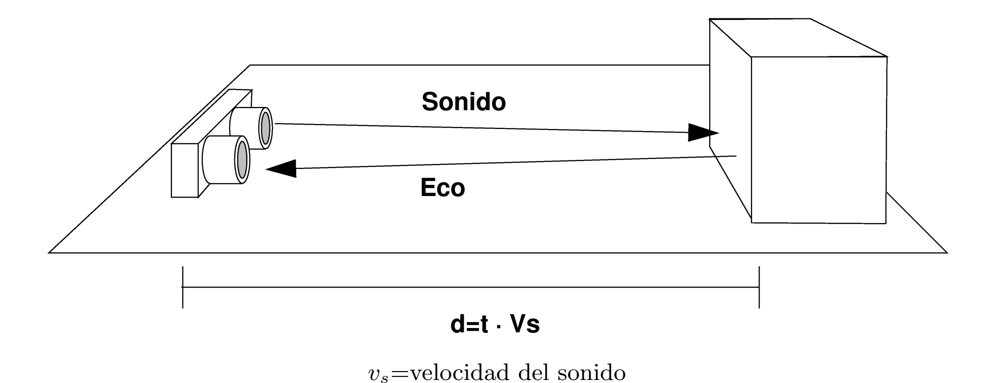
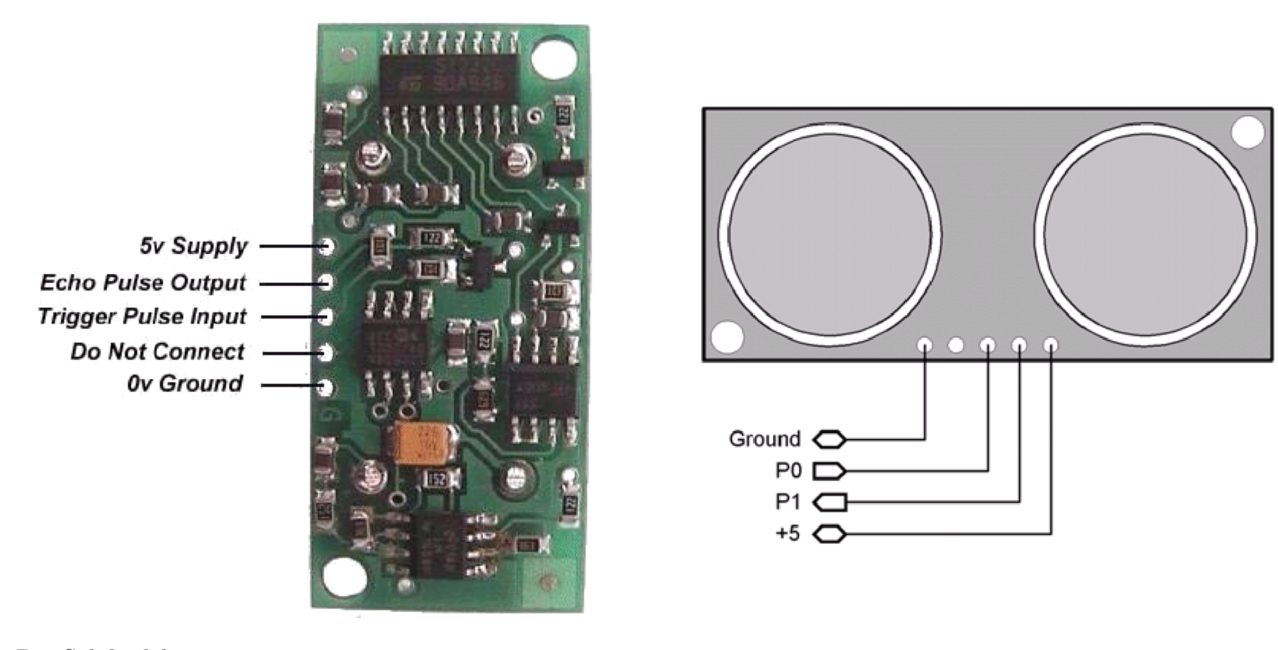


Figura 1: Esquema de medición de distancia

Las características de módulo son las siguientes:

- Alimentación: 3.5-5 [V].
- Consumo: Típ. 30 [mA], Máx. 50 [mA].
- Frecuencia: 40[KHz].
- Rango 3 [cm]-3 [m].

El diagrama de conexiones se muestra en la Figura 2.



$P1$=Salida del eco

$P0$= Entrada del Trigger

Figura 2: Foto del módulo (visión trasera) y Diagrama de conexiones (visión delantera)

Para hacer funcionar el módulo se debe poner un canto de bajada por la entrada $P1$. Luego el módulo genera el pulso sonoro que sale por el buzzer ultrasónico. Después de hecho esto pone en alto el pin $P0$, y lo baja al recibir el primer eco. En caso de no recibir eco alguno, se baja automáticamente el pulso luego de $36[ms]$ (en realidad es una especie de *timeout* para mediciones que exceden del rango).

Éstas formas de onda se muestran en la Figura 3.

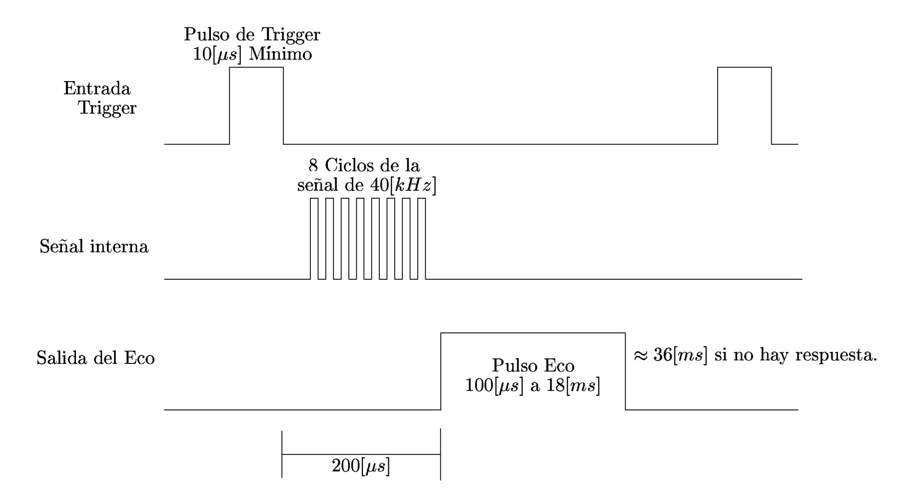


Figura 3: Formas de onda del módulo

El diseño del driver se basa en poder formar las ondas necesarias para la señal de trigger del módulo y *medir* el tiempo de la señal de respuesta para obtener la distancia.

3 Funcionamiento del driver

Como se mencionó el Driver consiste en crear las forma de onda que requiere el módulo y, luego, medir el ancho del pulso del eco para obtener finalmente la distancia, multiplicando la mitad de este tiempo por la velocidad del sonido.

Se decidió que la mejor forma de implementar las ideas mencionadas era utilizando un Timer para generar las formas y una Interrupción para medir el tiempo del eco.

Se debió elegir un pin con posibilidad de captura como entrada, se escogió utilizar el puerto FREQ (P1.0), ya que esta disponible en la tarjeta de desarrollo.

También por comodidad se decidió utilizar el pin A0 (P6.0) como salida para el trigger.

Entonces la realización del driver se reduce a coordinar el disparo del trigger con la captura del tiempo del eco.

Un diagrama de bloques simple que muestra el funcionamiento general es el siguiente:

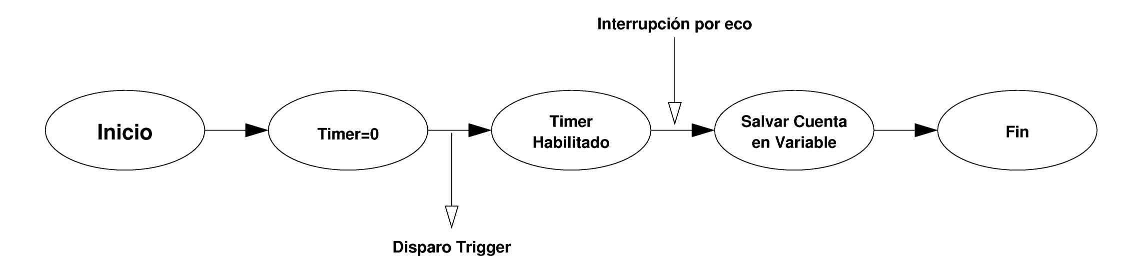


Figura 4: Diagrama de bloques del funcionamiento

Se consideró que sería bueno implementar dos modos de adquisición, uno que responda en llamada, denominado *Modo Único*, y un modo que este continuamente actualizando el valor de la medición, denominado *Modo Repetitivo*.

3.1 Modos de adquisión

El funcionamiento de cada modo es similar, ya que para ambos es necesario usar un Timer y la interrupción de una compuerta digital del microcontrolador, a continuación se detallará el funcionamiento de cada modo por separado.

3.1.1 Modo Adquisición única

En este modo se desea adquirir el valor de la distancia una sola vez. Al llamar la función que implementa el driver, ésta devuelve la medición al terminar de adquirirla (hasta 36 [ms]).

Funcionamiento:

Una vez llamada la función se realizan las siguientes operaciones:

i. Setea pin P6.0=1, configura la interrupción de la puerta digital y configura el Timer A.

ii. Espera 20[μs], genera trigger del sensor ultrasónico, bajando el pin de salida (P6.0=0). Setea en cero el registro de cuenta del Timer A.

iii. Se espera interrupción de la compuerta digital (pin P1.0).

iv. Al generarse la interrupción (P1.0=0),deshabilita la interrupción de ese pin y almacena el valor de la cuenta del Timer A en una variable.

v. Convierte y devuelve el valor de la variable en milímetros.

A continuación se presentan el diagrama de bloques y formas de onda del modo Adquisición única:

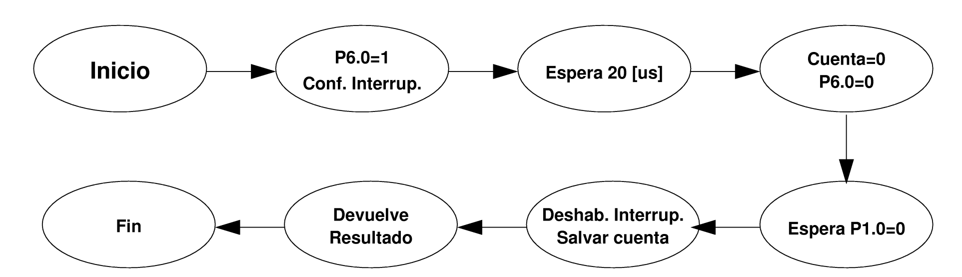


Figura 5: Diagrama de bloques Modo Único

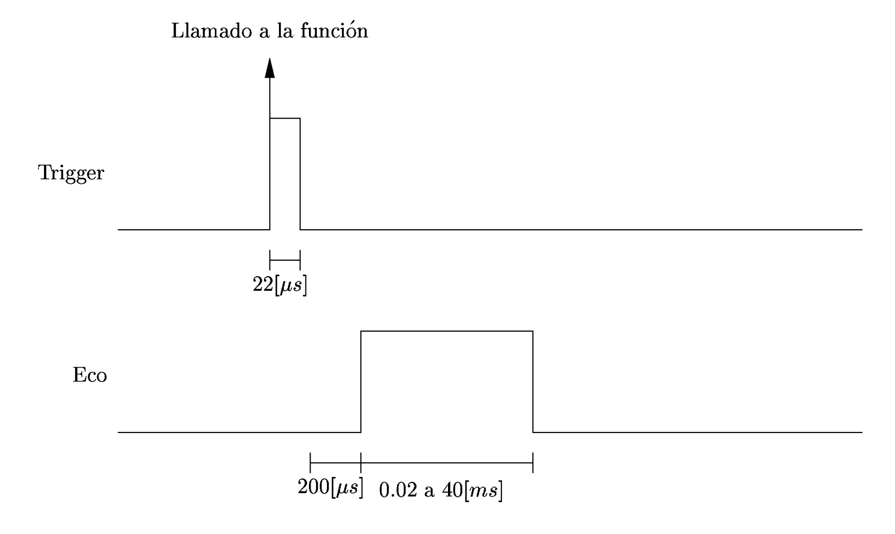


Figura 6: Formas de onda modo Único

3.1.2 Modo Adquisición repetitiva

La idea detrás de este modo es el tener al driver adquiriendo continuamente. Se llama una sola vez y queda funcionando para seguir ejecutando todo el código.

El modo repetitivo tiene dos formas de entregar la adquisición: la última adquisición y el promedio de las últimas 4 mediciones. La devolución de las mediciones se hace mediante variables globales.

La adquisición se realiza periódicamente, según se produce el *timer overflow* del Timer A.

Funcionamiento

En este modo se realizan las siguientes operaciones:

i. Setea pin P6.0=1, configura la interrupción del puerto digital y Timer Overflow del Timer A.

ii. Espera $20[\mu s]$, genera trigger del sensor ultrasónico, bajando el pin de salida. Setea en cero el registro de cuenta del Timer A.

iii. Al generarse la interrupción (P1.0=0), almacena el valor de la cuenta del Timer A en una variable. Calcula promedio de las últimas cuatro mediciones.

iv. Con la interrupción de *Timer Overflow* se repite el proceso desde i.

Se debe destacar que el periodo en que se toman las mediciones esta dado por el timer overflow.

A continuación se presentan el diagrama de bloques y formas de onda del modo Adquisición repetitiva:

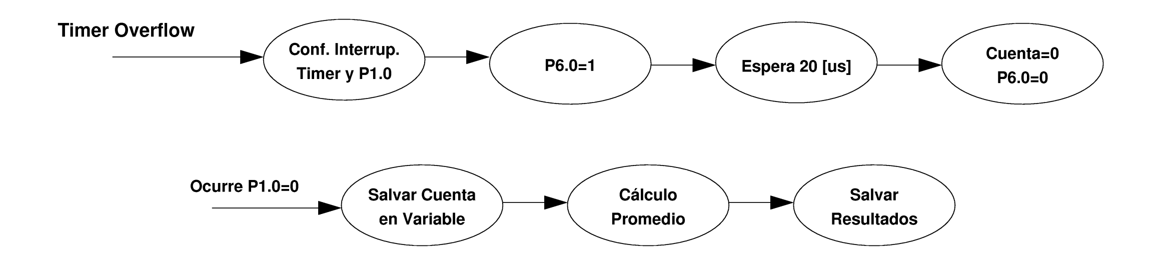


Figura 7: Diagrama de bloques Modo Repetitivo

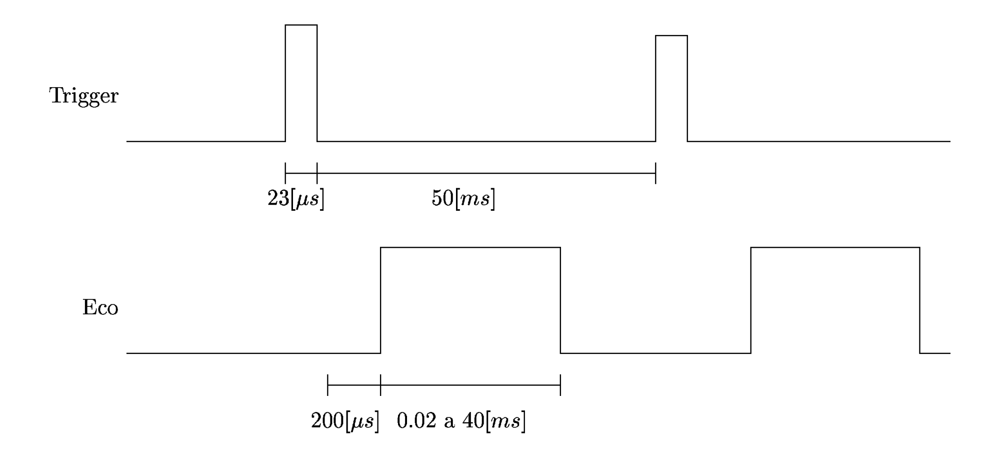


Figura 8: Formas de onda de Modo Repetitivo

Los detalles más operativos del funcionamiento se pueden ver en el código en la Sección 5

4 Manual del Usuario

Esta sección tiene por objeto definir como utilizar el driver.

4.1 Conexiones del módulo de ultrasonido:

El módulo puede ser alimentado con el voltaje V_{cc} ($+3.6[V]$) y GND ($0[V]$)de la tarjeta de desarrollo EasyWeb (para el diagrama de conexiones referirse a la Figura 2). De esta forma no se necesita fuente de alimentación externa.

El pin de trigger del módulo de ultrasonido se conecta al pin A0 de la tarjeta (P6.0). Por este pin se envían las señales desde la tarjeta EasyWeb hacia el sensor para iniciar las mediciones.

Además el pin de salida del módulo se conecta al pin FREQ de la tarjeta de desarrollo (P1.0). Mediante este pin se reciben los resultados de la medición del sensor que serán procesados por el driver.

4.2 Utilización del driver:

El código del driver está contenido en el archivo `drv_us.c`. Éste se debe agregar al programa utilizando una instrucción `#include "drv_us.c"` en el codigo.

El driver implementa dos funciones (de acuerdo a los modos descritos anteriormente): `Distance1()` para adquisición única y `Distance0()` para adquisición repetitiva.

Además, deben llamarse a las inicializaciones del driver con `InitUS()`, que inicializa los pines que utiliza, como entrada o salida según corresponda. La inicialización del Timer A y del pin de captura se realizan internamente.

Para utilizar la función de adquisición única, se llama sin parámetros, y retorna el valor de la distancia medida en milimetros (que se miden desde el 0 en el borde exterior de los buzzers). En caso de que la distancia exceda el máximo medible (3 metros), el driver retornará un 0.

Ejemplo:

```
int medida;
InitUS();
medida=Distance1();
printf("Distancia :%i",medida);
```

La función para medir repetidamente también se invoca sin parámetros. Esta función no retorna valor alguno, ya que se mantiene permanentemente realizando mediciones. Los resultados de dichas mediciones se puede acceder mediante las variables: `DIST` y `DIST_PROM`. En éstas se almacenan el último valor medido, y un promedio de las últimas cuatro mediciones.

Ejemplo:

```
InitUS();
Distance0();
while(1){
printf("Distancia: %i",DIST;
printf("Distancia Promedio: %i",DIST_PROM);}
```

Debe recordarse que en este modo se producirá una interrupción periódica en la ejecución del programa.

El driver no esta pensado para utilizar ambos modos al mismo tiempo.

5 Código

En esta sección están los códigos necesarios para utilizar el driver, también se incluye un programa de ejemplo que utiliza el driver y muestra la distancia por el LCD.

5.1 Código del driver

Se incluye a continuación el código del driver en sí.

drv_us.c

```
/*
Driver para módulo utrasonido SRF04
Pin de trigger se conecta al pin FREQ de la tarjeta de desarrollo (P1.0)
Pin de salida del módulo se conecta al pin P6.0
Resultado de la última conversión se obtiene en la variable DIST
Promedio de las últimas 4 conversiones se encuentra en al variable DIST_PROM
*/
#define trigger_on ( P6OUT |= BIT0 )    //Trigger para el módulo utrasonido.
#define trigger_off ( P6OUT &= ~BIT0 )

int Distance1 (void);
void Distance0 (void);
void InitTimer_A (void);
void Delay_US (unsigned int retardo);

unsigned int DISTANCIA[]={0,0,0,0};//Variables donde se guarda el
                         //resultado de las 4 últimas mediciones
long int DIST_SUM = 0;
unsigned int LISTO=1, DIST_PROM, DIST;

//****************************************************************************
void InitUS(void)
    {
    P6SEL &= ~BIT0;   // Pin 6.0 como I/O
    P6DIR |= BIT0;    // Pin 6.0 como salida.
    P1SEL &= ~BIT0;
    P1DIR &= ~BIT0;   //Pin Freq (P1.0) de la tarjeta se usa como entrada
    P1IES |= BIT0;    //Interrupción por canto de bajada
    }
```

```
#pragma vector=PORT1_VECTOR
__interrupt void Captura (void)
  {
  if (TAR < 25000)
    {
    DIST_SUM=0;
    DISTANCIA[3]=DISTANCIA[2];
    DIST_SUM += DISTANCIA[3];
    DISTANCIA[2]=DISTANCIA[1];
    DIST_SUM += DISTANCIA[2];
    DISTANCIA[1]=DISTANCIA[0];
    DIST_SUM += DISTANCIA[1];
    DISTANCIA[0] = TAR;        //almacenar la cuenta de Timer A
    DIST_SUM += DISTANCIA[0];      //en la variable DISTANCIA[0]
    DIST_PROM = (int)(DIST_SUM/4);
    DIST_PROM =(int)( ( ((long)(DIST_PROM) * 17) - 4538 )/100);
    DIST = (int)(( ((long)(DISTANCIA[0]) * 17) - 4538 )/100);
    }
  else
    {
    DISTANCIA[0] = 0;
    DIST=0;
    DIST_PROM=0;
    }
  LISTO=1;         //permite que salga de la funcion distance
  P1IFG &= ~BIT0;  //setear flag en 0 pq ya fue atendida la interrupcion
  }
//****************************************************************************

#pragma vector=TIMERA1_VECTOR
__interrupt void ClockHandler (void)
  {
  if (TAIV == 10)
    Distance0();    //Efectúa nueva medición
  }
//****************************************************************************

int Distance1 (void)     // modo único
  {
```

```
  trigger_on;
  LISTO=0;        //LISTO nos dice cuando vamos a obtener el valor de la distancia
  Delay_US(20);
  InitTimer_A();
  TAR = 0x00;        //TIMER A=0
  P1IE |= BIT0;     //Habilitamos interrupciones del pin P1.0
  P1IFG &= ~BIT0;
  _EINT();          // habilitación general int's.
  trigger_off;
  LISTO=0;
  while (LISTO==0); //no sale de la rutina hasta que LISTO sea uno.
  P1IE &= ~BIT0;  //En el caso que sea modo unico LISTO se iguala a 1 en la
  //rutina de interrupción de la puerta 1, sino LISTO = 1 siempre
  //se deshabilita interrupcion de la PUERTA 1 si el modo de operación es M=1
  return (DIST);
  }
void Distance0 (void)            // modo continuo
  {
  trigger_on;     //P6.0 = Trigger US = 1
  LISTO=0;    //LISTO nos dice cuando vamos a obtener el valor de la distancia
  Delay_US(20);
  InitTimer_A();
  TAR = 0;
  P1IE |= BIT0; //Habilitamos interrupciones del pin P1.0
  P1IFG &= ~BIT0;
  TACTL |= TAIE;
  _EINT();      // habilitación general int's.
  TAR = 0x0;    //TIMER A=0
  trigger_off;    //P6.0 = 0, implica trigger para el US
  }
//*******************************************************************

void InitTimer_A (void)
  {                                  //ACLK es 8MHz/2= 4MHZ
                                     //1[mm]=3+3[uS]=6[uS]
  TACTL = TASSEL_1 | ID_2 | MC_1;    //TAclk = ACLK & /8, cuenta hasta TACCR0,
                                     //Interrupción habilitada
  TACCR0 = 50000;                    //CUENTA HASTA 45.000 = 450[ms] = echo timeout
                                     //Si no detecta objeto, T= 42811
```

```
  }
void Delay_US (unsigned int retardo)
  {
  unsigned int j; //se define j local.
  for (j=0 ; j<retardo; j++);
  }
```

5.2 Programa de ejemplo

El código de ejemplo muestra la utilización del driver para visualizar la distancia medida en el display. Se pueden utilizar los modos de adquisición.

distancia.c

```
#include <msp430x14x.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "display_LCD.c"
#include "drv_us.c"

//Acá se selecciona que modo se desea probar.
//0 para continuo y 1 para único
#define MODO 1

void InitOsc(void);

int fi=0;   //Ennumera cuantas mediciones se han hecho
//**************************************************************************
//Programa Principal
void main(void)
  {
  int BOTON;
  WDTCTL = WDTPW | WDTHOLD;  //DETIENE WATCHDOG
  InitOsc();
  InitUS();
  InitLCD();
```

```
  printf("    Sensor ");
  SEND_CMD(LINEA2);
  printf("  Ultrasonico ");

//****** INICIO Ejemplo Modo Continuo *************************************
#if MODO == 0
  Distance0();      //se ocupa para probar modo continuo (0)
  while(1)
    {
    SEND_CMD(LINEA1);
    if(DIST >0)
        printf("Actual  : %d[mm]    ",DIST);    //Muestra la última medición
    else
        printf("Fuera Rango     ");
    SEND_CMD(LINEA2);
    printf("Promedio: %d[mm]     ",DIST_PROM);  //Muestra el promedio
    fi=fi+1;                                    //de las 4 últimas mediciones
    Delayx100us(1000);
    }
#endif
//****** FIN Ejemplo Modo Continuo ******************************************


#if MODO == 1

//****** INICIO Ejemplo Modo Único ******************************************
  while(1)
    {
    long DIST;
    DIST = (long)((DIST_PROM * 0.170) - 45.382);
    if (BOTON == 0)
    {
        BOTON = ((~P4IN & 0xF0)/16);
    if( BOTON == 1 )
      {
      DIST=Distance1();
      SEND_CMD(LINEA1);
      if(DIST >0)
        printf("%d[mm]        ",DIST);  //Muestra la última medición
```

```
       else
           printf("Fuera Rango     ");
       SEND_CMD(LINEA2);
           printf("Medicion N.%d   ",fi);
           fi=fi+1;
           }
     }
     else
     BOTON = ((~P4IN & 0xF0)/16);
    }
//****** FIN Ejemplo Modo Único *********************************************

#endif

  }
//Fin Programa Principal
//***************************************************************************

void InitOsc(void) {
  WDTCTL = WDTPW + WDTCNTCL;
  BCSCTL1 |= XTS;
  _BIC_SR(OSCOFF);
  do
    IFG1 &= ~OFIFG;
  while (IFG1 & OFIFG);

  BCSCTL1 |= DIVA0;
  BCSCTL1 &= ~DIVA1;
  IE1 &= ~WDTIE;
  IFG1 &= ~WDTIFG;
  WDTCTL = WDTPW | WDTTMSEL | WDTCNTCL | WDTSSEL | WDTIS1;
  while (!(IFG1 & WDTIFG));
  IFG1 &= ~OFIFG;
  BCSCTL2 |= SELM0 | SELM1;
}
//***********************************************************
```

5.3 Código Adicional

En esta sección se encuentran los códigos del driver del display LCD, para utilizar con el programa de ejemplo o cualquier aplicación en general.

Se destaca que el driver habilita la función `printf` para el display del LCD de la tarjeta Easyweb.

def_display.h

```
#define Clear_Display 0x01 // 0 0 0 0 0 0 0 1
#define Return_Home 0x02 // 0 0 0 0 0 0 1 *
#define Set_Entry_Mode 0x04 // 0 0 0 0 0 1 I/D SH
#define Set_Display 0x08 // 0 0 0 0 1 D C B
#define Set_Cursor_and_Display_Shift 0x10 // 0 0 0 1 S/C R/L * *
#define Set_Function 0x20 // 0 0 1 DL N F * *
#define Set_CGRAM_Address 0x40 // 0 1 A5 A4 A3 A2 A1 A0
#define Set_DDRAM_Address 0x80 // 1 A6 A5 A4 A3 A2 A1 A0
#define _100us 131 //131 cycles *6 + 13 = 799 / 799*125ns = 99,875 us
#define _10us 11 //11 cycles * 6 + 13 = 79 / 81*125ns=9,875 us
#define retdata 55 // 43 us
#define retins 50 // 39 us
#define retclear 15 // 1,53 ms 1530 us
#define E_OFF P2OUT &= ~BIT3 // P2.3 = 0
#define E_ON P2OUT |= BIT3 // P2.3 = 1
#define RS_OFF P2OUT &= ~BIT3 // P2.2 = 0
#define RS_ON P2OUT |= BIT3 // P2.2 = 1
#define Data_Register 0x04
#define Ins_Register 0x00
#define Enable_Low 0x00
#define Enable_High 0x08
#define Decrement_Address 0x00 // . . . . . . . 0 .
#define Increment_Address 0x02 // . . . . . . . 1 .
#define Shift_Display_Off 0x00 // . . . . . . . . 0
#define Shift_Display_On 0x01 // . . . . . . . . 1
#define Display_Off 0x00 // . . . . . . 0 . .
#define Display_On 0x04 // . . . . . . 1 . .
#define Cursor_Off 0x00 // . . . . . . . 0 .
#define Cursor_On 0x02 // . . . . . . . 1 .
#define Blink_Off 0x00 // . . . . . . . . 0
#define Blink_On 0x01 // . . . . . . . . 1
```

```
#define Cursor 0x00 // . . . . . 0 . . .
#define Display_and_Cursor 0x08 // . . . . . 1 . . .
#define Left 0x00 // . . . . . . 0 . .
#define Right 0x04 // . . . . . . 1 . .
#define Data_Length_4 0x00 // . . . 0 . . . . .
#define Data_Length_8 0x10 // . . . 1 . . . . .
#define One_Display_Line 0x00 // . . . . . 0 . . .
#define Two_Display_Lines 0x08 // . . . . . 1 . . .
#define Font_5x7 0x00 // . . . . . . 0 . .
#define Font_5x10 0x04 // . . . . . . 1 . .
#define Line2_Offset 0x40
```

display_LCD.c

```
#include "def_display.h"
unsigned char TXData, RXData,i,j,k,temp,RX_flag,cntr,time_out;
#define LCD_Data         P2OUT
#define RS              2//P2.2
//Definiciones extra para LCD
#define DISP_ON         0x0c //LCD control constants
#define DISP_OFF        0x08
#define CLR_DISP        0x01
#define CUR_HOME        0x02
#define ENTRY_INC     0x06
#define E                 3     //P2.3

#define bitset(var,bitno) ((var) |= 1 << (bitno))
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))

#define     LINEA1     0x80
#define     LINEA2     0xc0

void Delay (unsigned int retardo)
{
    unsigned int j;     //se define j local.
    for (j=0 ; j<retardo; j++);
}
void Delayx100us(unsigned int b)
```

```
{
    int j;
    for (j=0; j!=b; ++j) Delay (_100us);
}
void _E(void)
{
    bitset(P2OUT,E);     //toggle E for LCD
    Delay(_10us);
    bitclr(P2OUT,E);
}
void Put_Val(unsigned char val)
{
    E_ON;
    P2OUT = (P2OUT & 0x0F)|( val & 0xF0); //sale nibble superior
    Delay(1); // tw ancho del pulso E. min de 400ns
    E_OFF;
    Delay(1); // tiempo de ciclo de E. min de 1400 ns
    E_ON;
    P2OUT = (P2OUT & 0x0F)|((val<<4)& 0xF0); //sale nibble inferior
    Delay(1); //2,375 us tw
    E_OFF;
    Delay(1); //th1
    RS_OFF;
}
void Put_Ins(unsigned char Ins)
{
    RS_OFF;
    Delay(1);
    Put_Val(Ins);
    Delay(retins); // 39us
}
void Init_Display(void)
{
    Delayx100us(700); //esperar 70 ms. Queda en modo 8 bits.
    E_ON;
    P2OUT = Set_Function+Data_Length_4+Enable_High;
    Delay(1);
    E_OFF;
    Delay(retins);//esperar cambio a modo 4 bits
```

```
    Put_Ins(Set_Function + Data_Length_4 + Two_Display_Lines + Font_5x7);
    //no basta una
    Put_Ins(Set_Function + Data_Length_4 + Two_Display_Lines + Font_5x7);
    //la segunda es correctamente aceptada
    Put_Ins(Set_Display + Display_On + Cursor_On + Blink_Off);
    Put_Ins(Clear_Display);Delayx100us(retclear);
    Put_Ins(Set_Entry_Mode + Increment_Address + Shift_Display_Off);
}
void Put_Data(unsigned char Ch)
{
    RS_ON;
    Delay(1);
    Put_Val( Ch);
    Delay(retdata);
}
void SEND_CHAR (unsigned char d)
{
    Delayx100us(5);                 //.5ms
    temp = d & 0xf0;    //get upper nibble
    LCD_Data &= 0x0f;
    LCD_Data |= temp;
    bitset(P2OUT,RS); //set LCD to data mode
    _E(); //toggle E for LCD
    temp = d & 0x0f;
    temp = temp << 4; //get down nibble
    LCD_Data &= 0x0f;
    LCD_Data |= temp;
    bitset(P2OUT,RS); //set LCD to data mode
    _E(); //toggle E for LCD
}
void SEND_CMD (unsigned char e)
{
    Delayx100us(10);                //10ms
    temp = e & 0xf0;    //get upper nibble
    LCD_Data &= 0x0f;
    LCD_Data |= temp; //send CMD to LCD
    bitclr(P2OUT,RS); //set LCD to CMD mode
    _E(); //toggle E for LCD
    temp = e & 0x0f;
```

```
        temp = temp << 4; //get down nibble
        LCD_Data &= 0x0f;
        LCD_Data |= temp;
        bitclr(P2OUT,RS); //set LCD to CMD mode
        _E(); //toggle E for LCD
    }
    void InitLCD(void)
    {
        P2SEL = 0;
        P2OUT = 0;
        P2DIR = ~BIT0; //only P2.0 is input
        bitclr(P2OUT,RS);
        Delayx100us(250); //Delay 100ms
        Delayx100us(250);
        Delayx100us(250);
        Delayx100us(250);
        LCD_Data |= BIT4 | BIT5; //D7-D4 = 0011
        LCD_Data &= ~BIT6 & ~BIT7;
        _E(); //toggle E for LCD
        Delayx100us(100);//10ms
        _E(); //toggle E for LCD
        Delayx100us(100);//10ms
        _E(); //toggle E for LCD
        Delayx100us(100); //10ms
        LCD_Data &= ~BIT4;
        _E(); //toggle E for LCD
        SEND_CMD(DISP_ON);
        SEND_CMD(CLR_DISP);
        SEND_CMD(CUR_HOME);
    }
    int putchar(int u)
    {
        SEND_CHAR((unsigned char) u);
        return 1;
    }
```